04_homework_linear_regression

November 18, 2017

1 Programming assignment 4: Linear regression

```
In [1]: import numpy as np

        from sklearn.datasets import load_boston
        from sklearn.model_selection import train_test_split
```

1.1 Your task

In this notebook code skeleton for performing linear regression is given. Your task is to complete the functions where required. You are only allowed to use built-in Python functions, as well as any `numpy` functions. No other libraries / imports are allowed.

1.2 Load and preprocess the data

I this assignment we will work with the Boston Housing Dataset. The data consists of 506 samples. Each sample represents a district in the city of Boston and has 13 features, such as crime rate or taxation level. The regression target is the median house price in the given district (in $1000's).

More details can be found here: http://lib.stat.cmu.edu/datasets/boston

```
In [2]: X , y = load_boston(return_X_y=True)

        # Add a vector of ones to the data matrix to absorb the bias term
        # (Recall slide #7 from the lecture)
        X = np.hstack([np.ones([X.shape[0], 1]), X])
        # From now on, D refers to the number of features in the AUGMENTED dataset (i.e. includi

        # Split into train and test
        test_size = 0.2
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_size)
```

1.3 Task 1: Fit standard linear regression

```
In [8]: def fit_least_squares(X, y):
            """Fit ordinary least squares model to the data.

            Parameters
```

```
    ----------
    X : array, shape [N, D]
        (Augmented) feature matrix.
    y : array, shape [N]
        Regression targets.

    Returns
    -------
    w : array, shape [D]
        Optimal regression coefficients (w[0] is the bias term).

    """
    w_best = np.dot(np.dot(np.linalg.inv(np.dot(X.T, X)), X.T), y)
    return w_best
```

1.4 Task 2: Fit ridge regression

```
In [10]: def fit_ridge(X, y, reg_strength):
             """Fit ridge regression model to the data.

             Parameters
             ----------
             X : array, shape [N, D]
                 (Augmented) feature matrix.
             y : array, shape [N]
                 Regression targets.
             reg_strength : float
                 L2 regularization strength (denoted by lambda in the lecture)

             Returns
             -------
             w : array, shape [D]
                 Optimal regression coefficients (w[0] is the bias term).

             """
             N, D = X.shape
             w_best = np.dot(np.dot(np.linalg.inv(np.dot(X.T, X) + reg_strength * np.eye(D)), X.
             return w_best
```

1.5 Task 3: Generate predictions for new data

```
In [11]: def predict_linear_model(X, w):
             """Generate predictions for the given samples.

             Parameters
             ----------
             X : array, shape [N, D]
                 (Augmented) feature matrix.
```

```
        w : array, shape [D]
            Regression coefficients.

        Returns
        -------
        y_pred : array, shape [N]
            Predicted regression targets for the input data.

        """
        preds = np.dot(X, w)
        return preds
```

1.6 Task 4: Mean squared error

```
In [12]: def mean_squared_error(y_true, y_pred):
             """Compute mean squared error between true and predicted regression targets.

             Reference: `https://en.wikipedia.org/wiki/Mean_squared_error`

             Parameters
             ----------
             y_true : array
                 True regression targets.
             y_pred : array
                 Predicted regression targets.

             Returns
             -------
             mse : float
                 Mean squared error.

             """
             N = y_true.shape[0]
             diff = y_pred - y_true
             MSE = (1.0/N) * np.dot(diff.T, diff)
             return MSE
```

1.7 Compare the two models

The reference implementation produces * MSE for Least squares ≈ **23.98** * MSE for Ridge regression ≈ **21.05**

You results might be slightly (i.e. ±1%) different from the reference soultion due to numerical reasons.

```
In [13]: # Load the data
         np.random.seed(1234)
         X , y = load_boston(return_X_y=True)
         X = np.hstack([np.ones([X.shape[0], 1]), X])
```

```python
test_size = 0.2
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_size)

# Ordinary least squares regression
w_ls = fit_least_squares(X_train, y_train)
y_pred_ls = predict_linear_model(X_test, w_ls)
mse_ls = mean_squared_error(y_test, y_pred_ls)
print('MSE for Least squares = {0}'.format(mse_ls))

# Ridge regression
reg_strength = 1
w_ridge = fit_ridge(X_train, y_train, reg_strength)
y_pred_ridge = predict_linear_model(X_test, w_ridge)
mse_ridge = mean_squared_error(y_test, y_pred_ridge)
print('MSE for Ridge regression = {0}'.format(mse_ridge))
```

```
MSE for Least squares = 23.9843076118
MSE for Ridge regression = 21.0514870338
```